05-26-00

HAMILTON, BROOK, SMITH & REYNOLDS, P.C.

A

05/25/00

UTILITY PATENT APPLICATION TRANSMITTAL *(Only for new nonprovisional applications under 37 C.F.R. 1.53(b))*	Attorney Docket No.	1958.2001-000
	First Named Inventor or Application Identifier	James E. Carey
	Express Mail Label No.	EL564266910US
Title of Invention	PARALLEL TASK SCHEDULING SYSTEM FOR COMPUTERS	

APPLICATION ELEMENTS
See MPEP chapter 600 concerning utility patent application contents.

ADDRESS TO: **Assistant Commissioner for Patents**
Box Patent Application
Washington, D.C. 20231

1. [X] Fee Transmittal Form
   (Submit an original, and a duplicate for fee processing)
2. [X] Specification **Total Pages 15**
   (preferred arrangement set forth below)
   - Descriptive title of the invention
   - Cross References to Related Applications
   - Statement Regarding Fed sponsored R & D
   - Reference to microfiche Appendix
   - Background of the Invention
   - Summary of the Invention
   - Brief Description of the Drawings
   - Detailed Description
   - Claim(s)
   - Abstract of the Disclosure
3. [X] Drawing(s) (35 U.S.C. 113) **Total Sheets 7**
   [ ] Formal [X] Informal
4. [X] Oath or Declaration/POA **Total Pages 2**
   a. [X] Newly executed (original or copy)
   b. [ ] Copy from a prior application (37 C.F.R 1.63(d))
   (for continuation/divisional with Box 17 completed)
   [NOTE Box 5 below]
   i. [ ] DELETION OF INVENTOR(S)
   Signed statement attached deleting inventor(s) named in the prior application, see 37 C.F.R. 1.63(d)(2) and 1.33(b).
5. [ ] Incorporation By Reference *(useable if Box 4b is checked)*
   The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein
6. [ ] Microfiche Computer Program *(Appendix)*
7. [ ] Nucleotide and/or Amino Acid Sequence Submission (if applicable, all necessary)
   a. [ ] Computer Readable Copy
   b. [ ] Paper Copy (identical to computer copy)
   [ ] Pages
   c. [ ] Statement verifying identity of above copies

ACCOMPANYING APPLICATION PARTS

8. [X] Assignment Papers (cover sheet & documents)
9. [ ] 37 C.F.R. 3.73(b) Statement *(when there is an assignee)* [ ] Power of Attorney
10. [ ] English Translation Document *(if applicable)*
11. [ ] Information Disclosure Statement (IDS)/PTO-1449 [ ] Copies of IDS Citations
12. [ ] Preliminary Amendment
13. [X] Return Receipt Postcard (MPEP 503) *(Should be specifically itemized)*
14. [ ] Small Entity Statement(s) [ ] Statement filed in prior application, status still proper and desired
15. [ ] Certified Copy of Priority Document(s) *(if foreign priority is claimed)*
16. [ ] Other: ____________

17. **If a CONTINUING APPLICATION,** check appropriate box and supply the requisite information:
[ ] Continuation [ ] Divisional [ ] Continuation-in-part (CIP) of prior application No.:
Prior application information: Examiner: Group Art Unit:

18. **CORRESPONDENCE ADDRESS**

NAME	Rodney D. Johnson, Esq.				
	HAMILTON, BROOK, SMITH & REYNOLDS, P.C.				
ADDRESS	Two Militia Drive				
CITY	Lexington	STATE	MA	ZIP CODE	02421-4799
COUNTRY	USA	TELEPHONE	(781) 861-6240	FAX	(781) 861-9540

Signature	Rodney D. Johnson	Date	May 25, 2000
Submitted by Typed or Printed Name	Rodney D. Johnson	Reg. Number	36,558

Rev September 23, 1999

Date: May 25, 2000 Express Mail Label No. EL564266910US

Inventor: James E. Carey
Attorney's Docket No.: 1958.2001-000

PARALLEL TASK SCHEDULING SYSTEM FOR COMPUTERS

BACKGROUND

In computers, application programs execute software instructions on a processor to perform work. Modern computers allow those instructions to be divided into discrete tasks for processing. In a multi-threaded computing environment, the tasks are assigned to multiple computing threads for processing. The threads perform the task and return results to the application program.

In a typical free-thread environment, any available thread can be used to process a task. Tasks are assigned to worker threads by provider threads executing thread manager instructions. There is no predefined relationship between a worker thread and a task or application.

Typically, the thread manager queues tasks into a single task queue. When a worker thread becomes available, the next task in the task queue is assigned to that worker thread on a first-in, first-out basis. On a busy system, the worker threads can all be busy at any particular time. As a result, new tasks cannot be immediately assigned to a thread. This causes the single queue to become populated with waiting tasks.

SUMMARY

One problem with using a single queue to feed tasks to a plurality of worker threads is that it results in lock contention. When a worker thread becomes available, the queue is locked until the first waiting task is located and assigned to the thread. Subsequently freed threads must wait on that lock before proceeding, causing a

bottleneck. In a busy system, the free - but idling - worker threads can become a wasted resource because they spend a considerable amount of time in a wait state.

One solution to the problem is to define a task space as a plurality of task queues. Each task queue can then be associated with a respective worker thread. This approach can diminish lock contention problems because a free worker thread would generally only cause its own task queue to be locked. Other subsequently freed worker threads could continue to process tasks from their own task queues.

Such parallel task queues can use a task scheduling algorithm to distribute tasks amongst the queues. To obtain an even distribution, a random number generator can be employed to select a task queue. Although the randomly selected queue may be busy, it provides a starting point for locating an empty queue. Once an empty queue is located, the new task is placed on that queue for processing by the associated task.

While the randomization can evenly distribute the work, the task still may not be efficiently removed from its assigned queue. To reduce the waiting time of queued tasks, the task scheduling algorithm can include a method of stealing a queued task. In particular, a freed worker thread first checks its associated queue. If the queue is empty, the worker thread searches the other queues for a waiting task. That task can then be moved to the empty queue and processed by the worker thread.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects, features and advantages of the parallel task scheduling system for computers will be apparent from the following more particular description of embodiments, as illustrated in the accompanying drawings in which like reference characters refer to the same parts throughout the different views. The drawings are not necessarily to scale, emphasis instead being placed upon illustrating the principles of the invention.

FIG. 1 is a schematic block diagram of a client-server computing environment.

FIG. 2 is a block diagram of a prior art task scheduling system.

FIG. 3 is a schematic block diagram of a parallel task scheduling system.

FIG. 4 is a flowchart for a queue assignment method of the parallel task scheduler of FIG. 3.

FIG. 5 is a schematic block diagram of the system of FIG. 3 showing the assignment of a new task.

FIG. 6 is a flowchart of a task stealing method of the parallel task scheduler of FIG. 3.

FIG. 7 is a schematic diagram of the system of FIG. 3 showing the assignment of a stolen task.

DETAILED DESCRIPTION

FIG. 1 is a schematic block diagram of a client-server computing environment. In the environment 1, a client computer 10 communicates with a server computer 20 over a communications medium 5. The communications medium 5 may be any wired or wireless interconnection between the client 10 and server 20, including a direct network connection or a public switched communications network, such as the Internet. The server 20 can be a multi-processor 22 computer for accessing and manipulating data stored in a data store 29.

As shown, the client 10 is generally a single-processor 12 computer under the control of an operating system. The client 10 executes client application software 18 to perform work for a user. Some of that work may need to be handled by the server 20. In that event, a request is passed by the client 10 to the server 20.

The server 20 receives the request for processing by a server application 28. The software instructions that process that request have been assigned by the compiler to one or more discrete tasks. A task scheduling system 26 within the server operating system 24 is responsible for making sure each task is processed. After each task is completed, results are returned to the server application 28. Ultimately, the client request is filled and results are returned to the client application 18.

Although a task can be initiated on the server 20 by a client request, a task can be initiated by applications on the server 20 itself. Of course, the server application 28 can be simultaneously responding to requests from multiple clients 10. Problems with scheduling tasks become most acute when the server 20 is busy executing many tasks.

A particular embodiment is Oracle Express Server, version 6.3, commercially available from Oracle Corporation, Redwood Shores, California. In this embodiment, the server 20 accesses database data from the data store 9. Specifically, the database is a multidimensional database that can be simultaneously accessed by multiple users.

FIG. 2 is a block diagram of a prior art task scheduling system 26'. A thread manager 25' coordinates the processing of tasks. A plurality of worker threads W1....Wx are processing threads in a worker thread pool 30. A plurality of work tasks T1...TN are maintained on a task queue 40'. A provider thread P (possibly from a provider thread pool 35) executes thread manager 25' instructions to queue a task. When it receives a new task, the provider thread P acquires a mutually exclusive (Mutex) lock on the task queue 40'. The new task is then queued at the queue tail. The provider thread P then releases the lock. In this way, the provider thread P puts tasks waiting to be processed onto a single queue.

The worker threads W1...Wx remove all tasks from that single queue. When a worker thread (say Wx) is freed, it executes thread manager 25' instructions to acquire a task to execute. The worker thread Wx locks the task queue 40' through a Mutex lock and locates the task (say T3) at the head of the queue 40'. That task T3 is then assigned to the worker thread Wx. The worker thread Wx then releases the lock. The worker thread Wx processes the assigned task and returns the results.

The provider and worker threads use the Mutex lock to cooperatively manage the task queue 40'. While the queue is locked, freed worker threads, and any executing provider threads, must wait on the lock before accessing the task queue 40'. Although this technique maintains the first-come, first-served order of the task queue 40', the worker threads can collide trying to remove tasks from the task queue 40'. Because the task queue 40' is accessed in a single-threaded manner, the worker threads are forced to

serially access the queue 40'. This single-threaded access can cause a large amount of thread context switching and can be very inefficient.

FIG. 3 is a schematic block diagram of a parallel task scheduling system 26. As in FIG. 2, a plurality of worker threads W1...Wx are maintained in a worker thread pool 30. There are, however, a plurality of task queues Q1,...,Qx in a queue space 40. As illustrated, each task queue Q1...Qx is associated with a respective worker thread W1,...,Wx. Each task queue can store a plurality of individual tasks. A parallel task scheduler 25 manages the assignment of tasks to threads.

As illustrated, the first task queue Q1 has one queued task, T1. The second task queue Q2 is empty. The last task queue Qx has two queued tasks T2, Tm. Here, although the second worker thread W2 is free, the last worker thread Wx is overburdened.

Such bottlenecks can occur because not all tasks are of equal complexity. A worker thread that draws a complex task can have a populated task queue, while a worker thread that draws simple tasks has an empty task queue. As a result, the processing of a simple task can be delayed by a complex task. It is therefore still possible that the associated worker thread (e.g. Wx) is overburdened by a complex task (e.g. T2) and will not process the queued task (e.g. Tm) immediately.

To reduce bottlenecks, the task scheduling system 26 attempts to place a new task on an empty task queue, if one exists. In particular, the task scheduling algorithm uses a random number generator to identify an initial, seed queue. If that randomly selected queue is not empty, the algorithm marches through the queues - starting from the seed queue - until an empty queue is found. The task is then placed on that queue.

Just because an empty queue has been found, however, does not guarantee that the queued task will be processed quickly. The associated worker thread may still be busy processing a complex task. The queue task may have to wait for the processing task to finish. Also, depending on the implementation of the system 26 and the system configuration, the randomly selected queue may not be empty.

Because any worker thread is suitable for processing any task, the parallel task scheduling system 26 can take advantage of additional methods to increase performance. In particular, another method is used by freed worker threads to steal waiting tasks from busy queues.

Using the parallel queue approach, each task queue is primarily processed by the associated (or owning) worker thread, with periodic access from the task provider thread and infrequent access from the other worker threads as they finish their tasks. Because there are an increased number of locks controlling access to the queues and a decreased number of threads attempting to gain access to the queues, the process in much more efficient and scalable.

FIG. 4 is a flowchart for a queue assignment method of the parallel task scheduler of FIG. 3. The queue assignment method 50 addresses how new tasks are queued for processing. This method is executed by a provider thread P.

The method first selects a random queue at step 51. A pseudo-random number generator (PRNG) is used to pick a queue number, modulo x, when x is the count of task queues. It should be noted that the particular PRNG is not critical to operation of the method, so any convenient PRNG can be used.

At step 52, the selected queue is locked by the method and first examined to determine if it is an empty queue. If the queue is not empty, the lock may be released and the next queue selected at step 53 by incrementing the queue number (modulo x). This process can continue until an empty queue is found, at step 52. In addition, the search can be halted after a specific time-out condition has been met, such as a predetermined number of increments. In an appropriately configured system, however, an empty queue should be found with little searching.

In another embodiment of step 52, the provider thread "peeks" at the associated queue, without a lock, to see if the queue is in a busy state. This is done by looking at the queue without holding the Mutex. While the peeker cannot operate on the queue using the information retrieved from the peek, the answer to the question "Is the queue empty?" is valid. This check, while not guaranteed accurate, can be very fast and

allows worker threads that may be busy to be skipped with little penalty to the speed of the check.

The protocol described for the queuing task guarantees that a task deposited by the task provider will have an active thread if the queue belonged to a waiting thread. If the queue is not empty, then the matching worker must either be busy, or be about to remove the task. If there is a task queued, the Mutex is taken to be sure it is really there (not in the process of being removed); if there is no task there, we do not need to take the lock to be sure there is no task there.

Consequently, when a queue is found that does not appear busy, the controlling lock on the queue is taken. If the queue is really empty, the task is deposited on the associated queue. In a small number of cases, the queue may no longer be empty after the lock is taken. In that case, the lock is dropped and the search continues. It is important to note that this type of collision should happen infrequently - e.g., on a very busy server.

In any event, an empty queue will generally be found. At step 54, the task is placed on the selected queue and the method releases the lock. The worker thread associated with that queue should process the task. If the associated worker thread is busy processing a complex task, it may take a relatively long time for the worker thread to again access its queue. Unless dealt with, that scenario could reduce processing efficiency.

FIG. 5 is a schematic block diagram of the system of FIG. 3 showing the assignment of a new task. As shown, the task assignment method 40 (FIG. 4) has found the empty task queue Q2 (FIG. 3). As a result, the new task Tn has been added to that queue Q2 for processing by the associated worker thread W2.

FIG. 6 is a flowchart of a task stealing method of the parallel task scheduler of FIG. 3. The method 60 is initiated by a worker thread completing a task. In a particular embodiment, upon completing a task, a worker thread goes through a wake-up process to reinitialize its thread data and to grab a new task. One problem is that the associated queue can be empty, while other queues are populated. In general, such a situation may

arise only when there are more tasks available for processing than there are threads to process the tasks.

At step 61, the worker queue is examined. If the queue is populated with a task, then processing jumps to step 66 to process that task. If the queue is empty, however, the method 60 begins searching for a task queued in a queue for another worker thread.

The effort of finding another task begins at step 62, where, the next queue is selected. The selection can be made by simply incrementing the queue number, modulo x. Other techniques can also be used.

Processing then continues to step 63, where the selected queue is examined. If a task is queued, processing continues to step 65. Otherwise, processing continues to step 64.

At step 64, an optional time-out check can be made. In one embodiment, the check is based on a complete cycle through the queues. That is, if the queue number is incremented back to the worker's queue number, then processing can jump to step 67 to discontinue. The time-out could also be a predetermined number of increments. The time-out could also result from an interrupt resulting from a task being queued to the worker thread's previously empty queue. As another alternative, idle threads can continuously scan for stealable tasks. Until a time-out, processing returns to step 62 to select the next candidate queue.

Once a stealable task is found, the task is moved from the selected queue to the worker's queue at step 65. At step 66, the first task in the worker's queue is processed by the worker thread.

After completion of the task, or a timeout, the worker thread is placed into a sleep mode at step 67.

FIG. 7 is a schematic diagram of the system of FIG. 3 showing the assignment of a stolen task. As shown, the task stealing method 50 (FIG. 6) has been used by the free worker thread W2 to steal the waiting task Tm from the busy queue Qx. The stolen task Tm is now in the second queue Q2 for processing by the associated worker thread W2. That task Tm can now be more efficiently handled.

Those of ordinary skill in the art will recognize that methods involved in the parallel task scheduling system for computers may be embodied in a computer program product that includes a computer usable medium. For example, such a computer usable medium can include a readable memory device, such as a solid state memory device, a hard drive device, a CD-ROM, a DVD-ROM, or a computer diskette, having computer readable program code segments stored thereon. The computer readable medium can also include a communications or transmission medium, such as a bus or a communications link, either optical, wired, or wireless, having program code segments carried thereon as digital or analog data signals.

While this parallel task scheduling system for computers has been particularly shown and described with references to particular embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made therein without departing from the scope of the invention encompassed by the appended claims. For example, the methods of the invention can be applied to various environments, and are not limited to the environment described herein.

CLAIMS

What is claimed is:

1. In a multithreaded computing environment, a method of processing computing tasks, comprising:

    defining a plurality of worker threads, each thread capable of processing a task;

    defining a plurality of task queues, each task queue capable of queuing a plurality of tasks;

    associating each task queue with a respective worker thread; and

    assigning a task to a task queue in an essentially random fashion.

2. The method of Claim 1 wherein assigning a task comprises selecting an empty task queue.

3. The method of Claim 2 wherein selecting comprises determining whether the selected task queue is in a busy state.

4. The method of Claim 1 further comprising, from a worker thread, processing a task from the associated task queue.

5. The method of Claim 1 further comprising, from a worker thread, processing a task from a task queue not associated with the thread.

6. In a multithreaded computing environment, a method of processing computing threads, comprising:

    defining a plurality of worker threads, each thread capable of processing a task;

defining a plurality of task queues, each task queue capable of queuing a plurality of tasks accessible by the worker threads;

associating each task queue with a respective worker thread;

assigning a task to an assigned task queue; and

in a worker thread not associated with the assigned task queue, processing the task

7. The method of Claim 6 where assigning comprises selecting the assigned task queue based on an essentially random number.

8. The method of Claim 6 wherein assigning comprises selecting an empty task queue.

9. The method of Claim 8 wherein selecting comprises determining whether the task queue is in a busy state.

10. In a multithreaded computing environment, a system for processing tasks, comprising:

a plurality of worker threads, each thread capable of processing a task;

a plurality of task queues, each task queue capable of queuing a plurality of tasks and each task queue associated with a respective worker thread; and

a task scheduler for a task to a task queue in an essentially random fashion.

11. The system of Claim 10 wherein the task scheduler selects an empty task queue for assigning the task.

12. The system of Claim 11 wherein the task scheduler further determines whether the selected task queue is in a busy state.

13. The system of Claim 10 further comprising a worker thread processing a task from the associated task queue.

14. The system of Claim 10 further comprising a worker thread processing a task from a task queue not associated with the thread.

15. In a multithreaded computing environment, a system for processing computing threads, comprising:

a plurality of worker threads, each thread capable of processing a task;

a plurality of task queues, each task queue capable of queuing a plurality of tasks accessible by the worker threads and each task queue associated with a respective worker thread;

a task scheduler for assigning a task to an assigned task queue; and

wherein the assigned task is processed by a thread not associated with the assigned task queue.

16. The system of Claim 15 where the task scheduler selects the assigned task queue based on an essentially random number.

17. The system of Claim 15 wherein the task scheduler selects an empty task queue for assigning the task.

18. The system of Claim 17 wherein the task scheduler further determines whether the task queue is in a busy state.

19. An article of manufacturing, comprising:

a computer-readable medium;

a computer implemented program for processing computing tasks in a multithreaded computing environment embodied in the medium, the comprising instructions for:

defining a plurality of worker threads, each thread capable of processing a task;

defining a plurality of task queues, each task queue capable of queuing a plurality of tasks;

associating each task queue with a respective worker thread; and

assigning a task to a task queue in an essentially random fashion.

20. The article of Claim 19 wherein the instructions for assigning a task comprise selecting an empty task queue.

21. The article of Claim 20 wherein the instructions for selecting comprise determining whether the selected task queue is in a busy state.

22. The article of Claim 19 further comprising instructions for processing, in a worker thread, a task from the associated task queue.

23. The article of Claim 19 further comprising instructions for processing, in a worker thread, a task from a task queue not associated with the thread.

24. An article of manufacture, comprising:

a computer-readable medium;

a computer-implemented program for processing computing threads, in a multithreaded computing environment embodied in the medium, the program comprising instructions for:

defining a plurality of worker threads, each thread capable of processing a task;

defining a plurality of task queues, each task queue capable of queuing a plurality of tasks accessible by the worker threads;

associating each task queue with a respective worker thread;

assigning a task to an assigned task queue; and

in a worker thread not associated with the assigned task queue, processing the task

25. The article of Claim 24 where the instructions for assigning comprise selecting the assigned task queue based on an essentially random number.

26. The method of Claim 24 wherein the instructions assigning comprises selecting an empty task queue.

27. The method of Claim 26 wherein the instructions for selecting comprise determining whether the task queue is in a busy state.

PARALLEL TASK SCHEDULING SYSTEM FOR COMPUTERS

ABSTRACT OF THE DISCLOSURE

A parallel task scheduling system in a multi-threaded computing environment includes a plurality of parallel task queues. Each task queue is associated with a respective worker thread from a plurality of worker threads. Each new task is assigned to one of the task queues. That assignment process including selecting a random queue and, from that starting point, locating an empty queue (if one exists). The task is then placed on that empty queue for processing.

Typically, the worker thread associated with the identified task queue will process the queued task. If the worker thread is busy processing another task, the queued task may be stolen by a free thread. A waiting task, can thus be processed in an efficient manner.

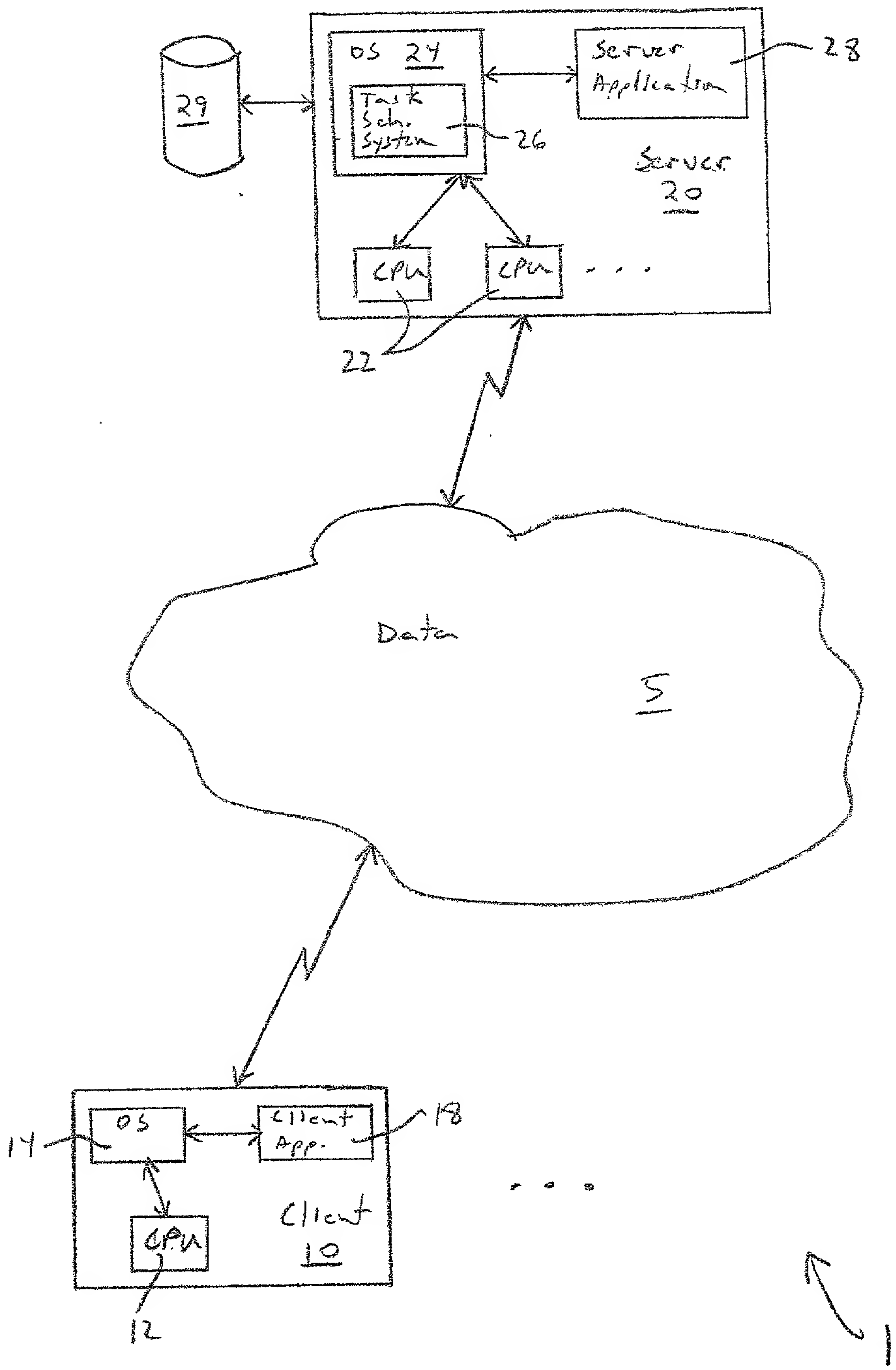
29
OS 24
Task Sch. System
26
Server Application
28
Server 20
CPU
CPU
22
Data
5
OS
14
Client App.
18
CPU
12
Client 10
1

FIG. 1

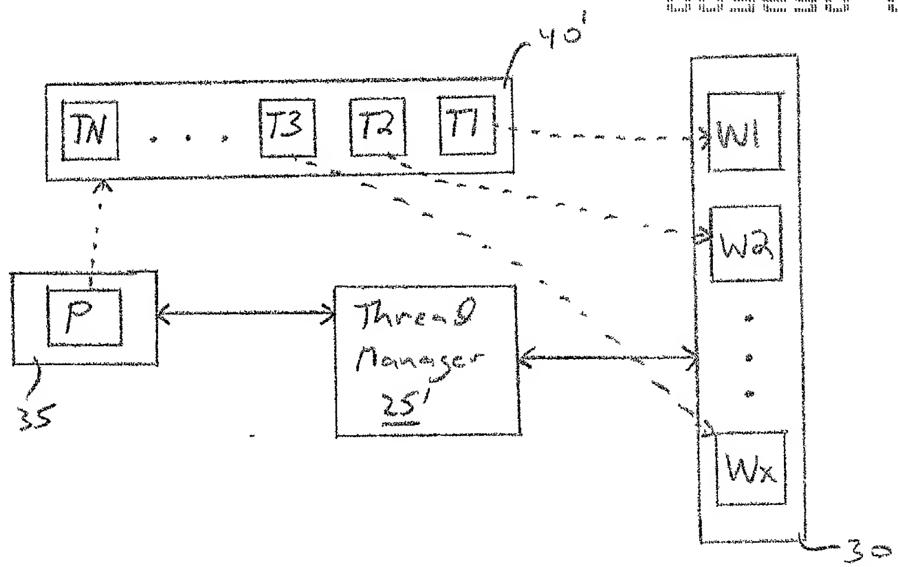

FIG. 2
(Prior Art)

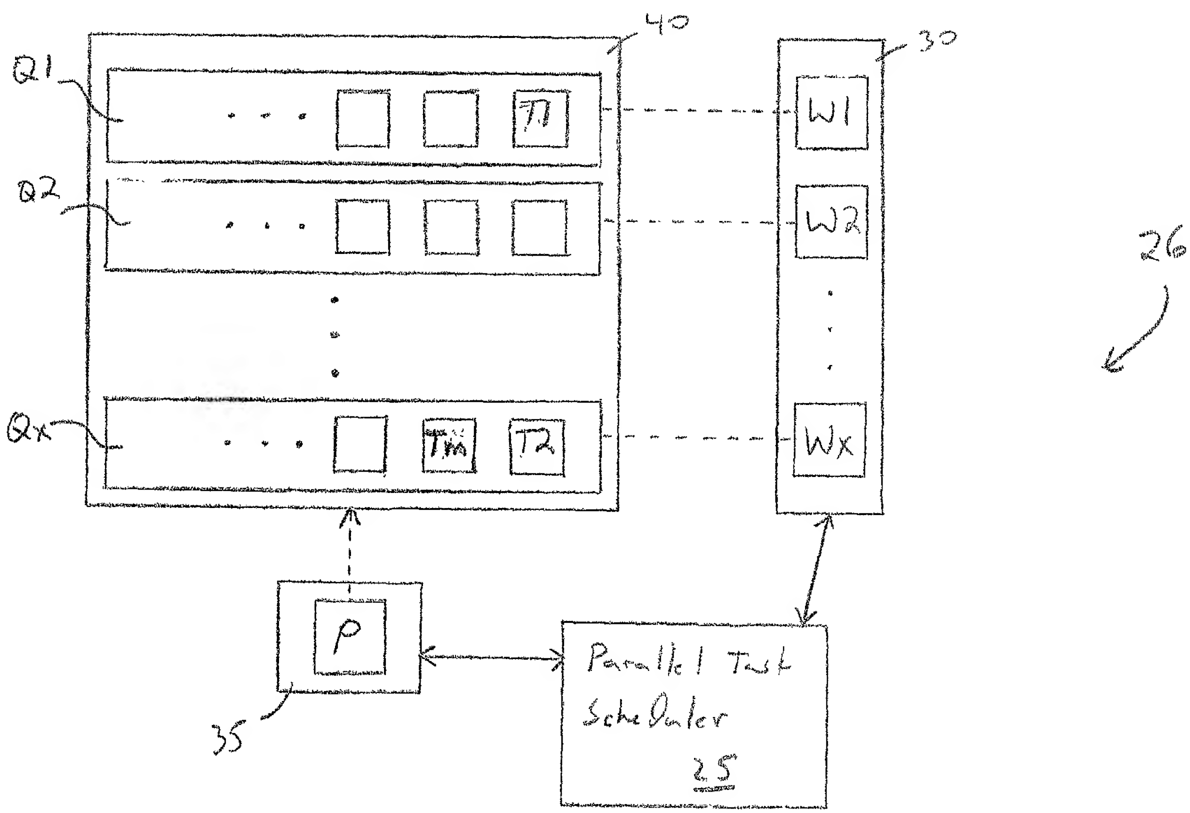
40
30
Q1
Q2
Qx
T1
W1
W2
Tm
T2
Wx
26
P
35
Parallel Task Scheduler
25

FIG. 3

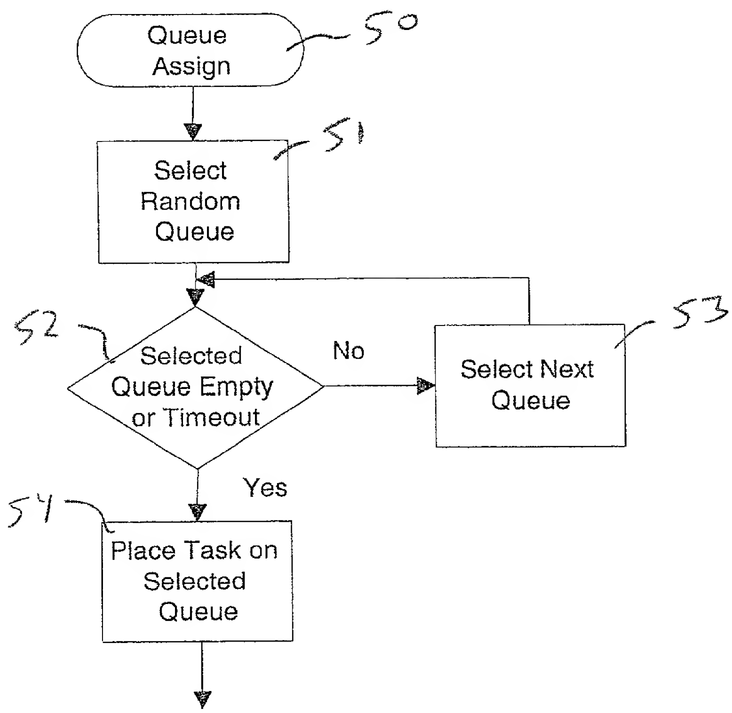
Queue Assign
50
Select Random Queue
51
52
Selected Queue Empty or Timeout
No
Select Next Queue
53
Yes
54
Place Task on Selected Queue

FIG. 4

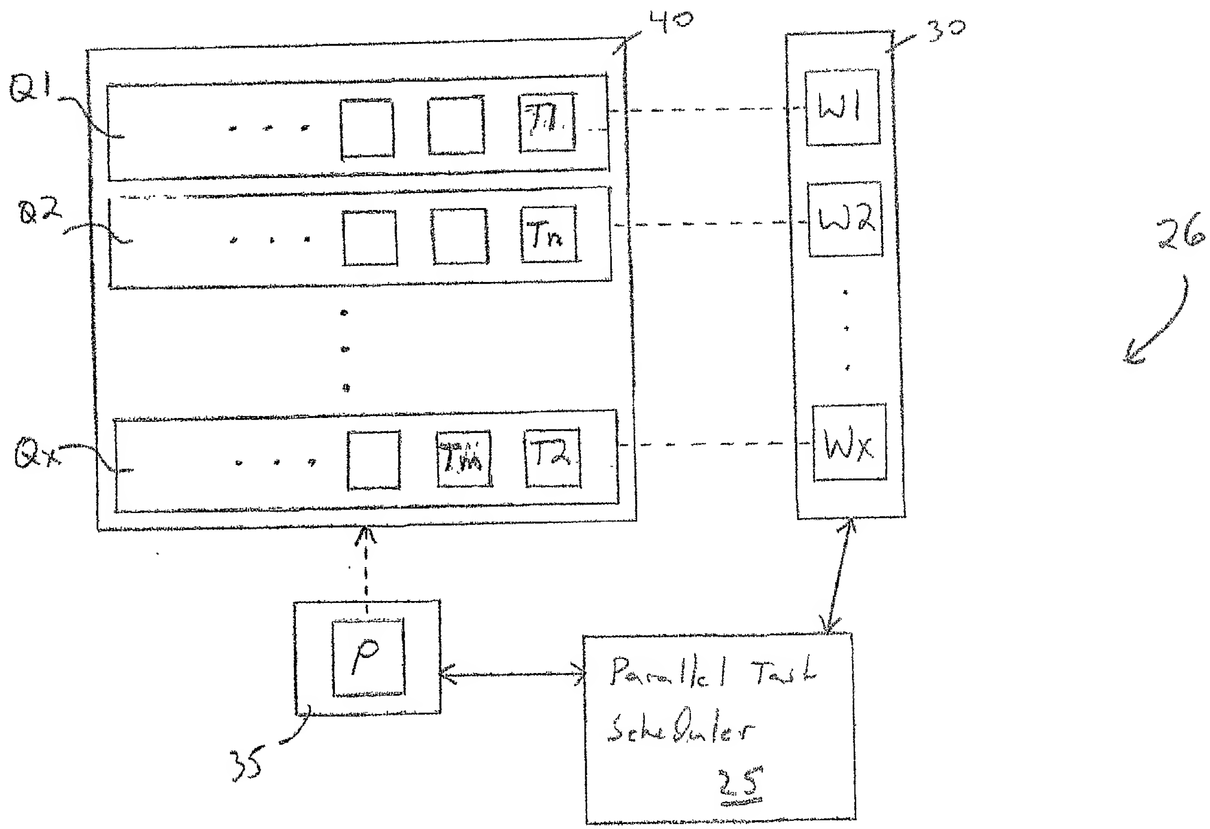
40
30
Q1
Q2
Qx
T1
Tn
Ti
T2
W1
W2
Wx
26
P
35
Parallel Task Scheduler
25

FIG. 5

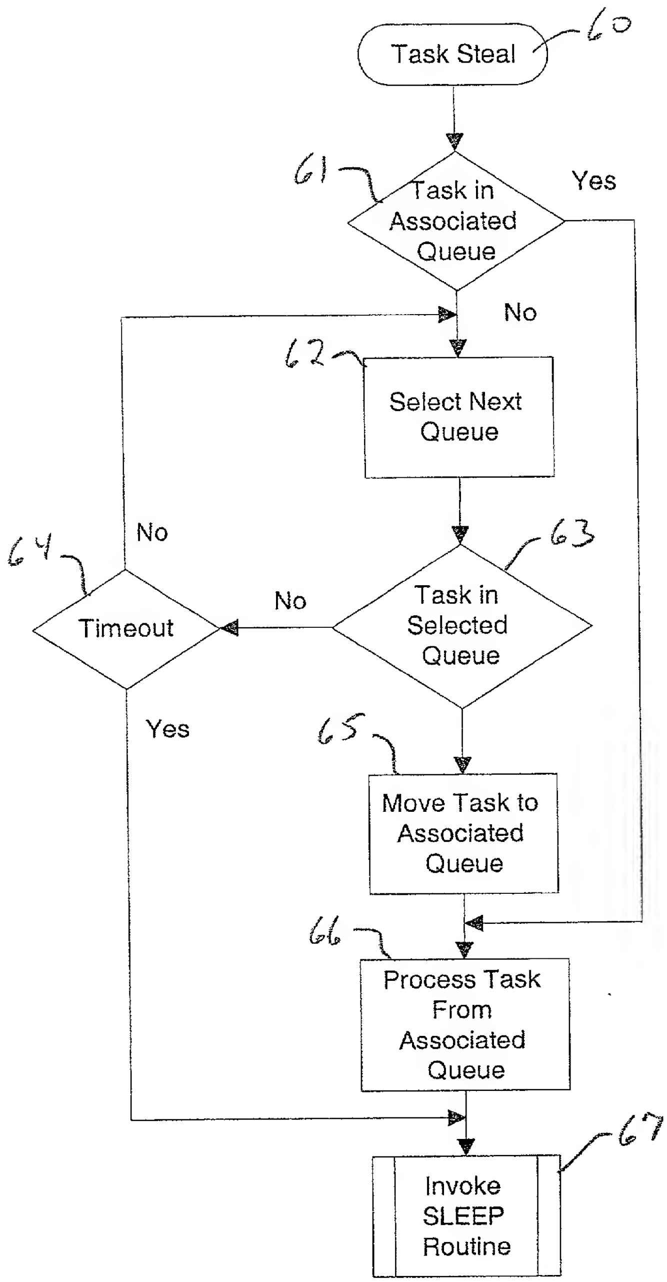
Task Steal
60
61
Task in Associated Queue
Yes
No
62
Select Next Queue
No
64
63
Timeout
No
Task in Selected Queue
Yes
65
Move Task to Associated Queue
66
Process Task From Associated Queue
67
Invoke SLEEP Routine

FIG. 6

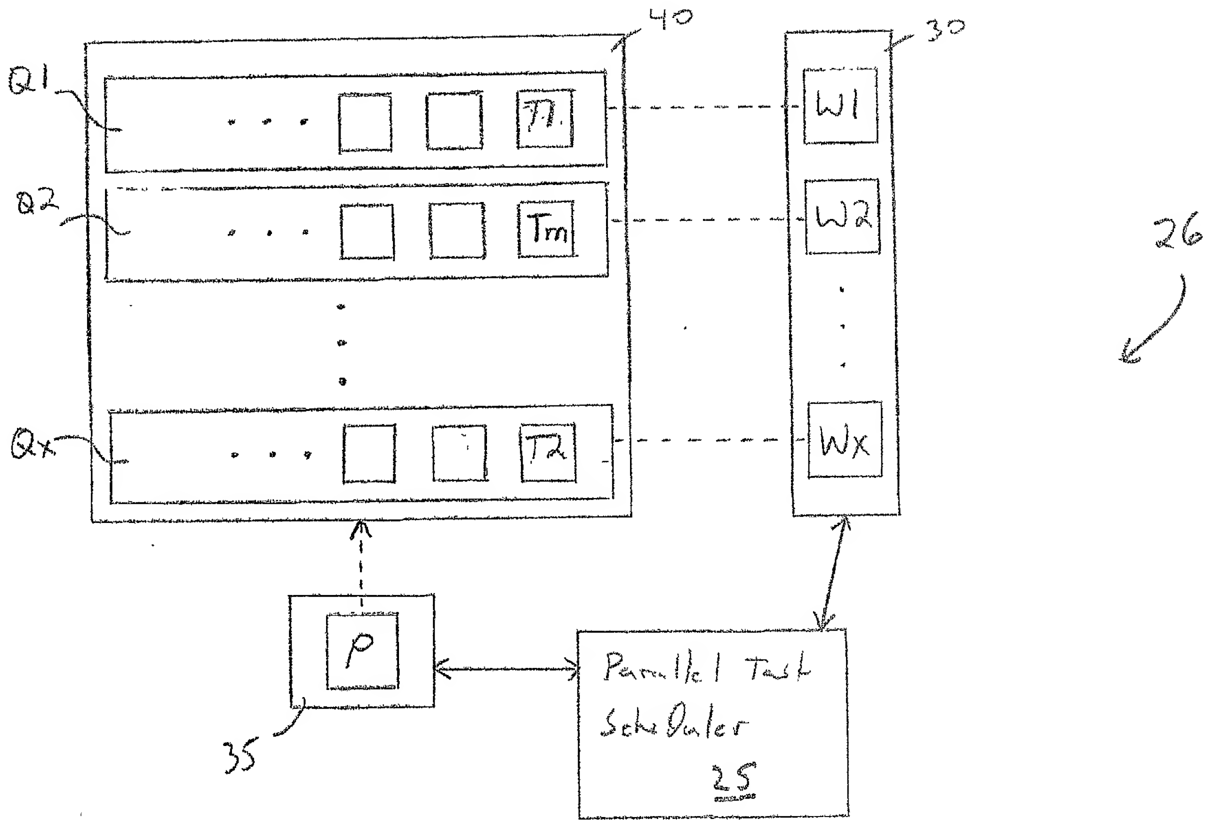
40
30
Q1
Q2
Qx
T1
Tm
T2
W1
W2
Wx
26
P
35
Parallel Task
Scheduler
25

FIG. 7

DOCKET NO. 1958.2001-000

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Declaration for Patent Application

As a named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated next to my name;

I believe I am the original, first and sole inventor (if only one name is listed) or an original, first and joint inventor (if plural names are listed in the signatory page(s) commencing at page 3 hereof) of the subject matter which is claimed and for which a patent is sought on the invention entitled

PARALLEL TASK SCHEDULING SYSTEM FOR COMPUTERS

the specification of which (check one)

[X] is attached hereto.

[ ] was filed on ______________ as United States Application

Number or PCT International Application No.______________

and was amended on ____________ (if applicable).

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is known by me to be material to patentability as defined in 37 C.F.R. §1.56.

I hereby claim foreign priority benefits under 35 U.S.C. 119(a)-(d) or 365(b) of any foreign application(s) for patent or inventor's certificate or 365(a) of any PCT international application which designated at least one country other than the United States of America, listed below and have also identified below, by checking the box, any foreign application for patent or inventor's certificate, or of any PCT international application having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s)			Priority Not Claimed	Certified Copy Filed? YES	NO
(Number)	(Country)	(Day/Month/Year filed)	[ ]	[ ]	[ ]
(Number)	(Country)	(Day/Month/Year filed)	[ ]	[ ]	[ ]
(Number)	(Country)	(Day/Month/Year filed)	[ ]	[ ]	[ ]

I hereby claim the benefit under 35 U.S.C. §119(e) of any United States provisional application(s) listed below.

(Application Number)	(Filing Date)
(Application Number)	(Filing Date)

I hereby claim the benefit under 35 U.S.C. 120 of any United States application(s), or 365(c) of any PCT international application designating the United States of America, listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States or PCT International application in the manner provided by the first paragraph of 35 U.S.C. 112, I acknowledge the duty to disclose information known by me to be material to patentability as defined in 37 C.F.R. 1.56 which became available between the filing date of the prior application and the national or PCT international filing date of this application:

(Application Serial No.)	(Filing date)	(Status: patented, pending, abandoned)
(Application Serial No.)	(Filing date)	(Status: patented, pending, abandoned)
(Application Serial No.)	(Filing date)	(Status: patented, pending, abandoned)
(Application Serial No.)	(Filing date)	(Status: patented, pending, abandoned)

As a named inventor, I hereby appoint the attorneys and/or agents associated with
Hamilton, Brook, Smith & Reynolds, P.C., Two Militia Drive, Lexington, Massachusetts 02421-4799
Customer No. 21005,

and ______________________________,

to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith.

Please send correspondence to:

[X] Customer No. **21005**
HAMILTON, BROOK, SMITH & REYNOLDS, P.C.
Two Militia Drive
Lexington, MA 02421-4799

or

[ ] **Address as follows:**

Direct telephone calls to: Rodney D. Johnson Telephone No.: 781-861-6240

Direct facsimiles to: Rodney D. Johnson Facsimile No.: 781-861-9540

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full name of sole or first inventor James E. Carey

Inventor's Signature James E Carey Date 5/25/2000

Residence 262 Winchester Street
Brookline, MA 02446

Citizenship US

Post Office Address (same as above)